Keil µVision and Actel SmartFusion V 2.0

by Robert Boys bob.boys@arm.com

Introduction:

This note describes the process of operating Keil µVision® and Actel's new SmartFusion family which contains an embedded ARM® Cortex™-M3 processor. SmartFusion contains the Serial Wire Viewer (SWV) debug technology that is available on Cortex-M*x* processors and supported by Keil. This note describes how to get all the components of this important technology working with µVision. **This article needs MDK® 4.11 or later.**

Keil µVision:

µVision is the IDE and is a component of the Keil MDK™ development system. Keil supports all 8051, ARM7, Cortex-M1 and Cortex-M3 processors as used in various Actel products.

MDK-ARM™ components include the µVision IDE, ARM Realview® compiler, assembler and RTX™ RTOS.

The Keil ULINK® JTAG/SWD adapter family includes the ULINK2, ULINK-ME and the ULINK*pro*.

RL-ARM™ contains the sources for RTX plus a TCP/IP stack a FLASH file system, USB and CAN drivers.

Why Use Keil MDK ?

MDK provides these features particularly suited for Actel users:

1. µVision IDE with Integrated Debugger, Flash programmer and the RealView ARM compiler.
2. A full feature RTOS is included with MDK: RTX by Keil. No royalty payments are required.
3. Serial Wire Viewer trace capability is included.
4. RTX Kernel Awareness window. It is updated in real-time and uses no system resources.
5. Choice of USB adapters: ULINK2, ULINK-ME, ULINK*pro* and Segger J-Link.
6. Kernel Awareness for Keil RTX, CMX, Quadros and Micrium. All RTOSs will compile with MDK.
7. Keil Technical Support is included for one year. This helps you get your project completed faster.

Serial Wire Viewer:

Serial Wire Viewer (SWV) displays PC Samples, Exceptions (including interrupts), data reads and writes, ITM, CPU counters and a timestamp. This information comes from the ARM CoreSight™ debug module integrated into the SmartFusion Cortex-M3. SWV does not steal any CPU cycles and is non-intrusive.

This document details these features:

1. Serial Wire Viewer (SWV).
2. Real-time Read and Write to memory locations for Watch, Memory and RTX Tasks windows.
3. Breakpoints and Watchpoints (Access Breaks).
4. RTX Viewer: a kernel awareness program for the Keil RTOS – RTX.
5. Keil TCP/IP stack – is one component of RL-ARM. A http server example is described on page 28.

Keil Contact Information: www.keil.com

USA: North America:
Keil, An ARM Company
Plano, Texas
800-348-8051 (Toll Free)
sales.us@keil.com
support.us@keil.com

Europe and Asia:
Keil, An ARM Company
Grasbrunn, Germany
+49 89/456040-20
sales.intl@keil.com
support.intl@keil.com

Copyright © 2010 ARM Ltd. All rights reserved

INDEX: **Page:**

Part 1: Introduction and Getting Ready: 3

1) Serial Wire Summary: 3
2) What the trace is useful for: 3
3) What kind of data can the Serial Wire Viewer see ? 3
4) Hardware: The Actel boards and debug adapters: 4
5) Software: Keil MDK 4.1: 5
6) Connecting the Hardware: 5

Part 2: Tracing Examples and Useful Information: 6

1) The Keil Blinky example: 6

Part 3: Configuring the Serial Wire Trace: 6

1) Debug Ports: JTAG, SWD, SWJ and SWO. A short tutorial: 6
2) Selecting SWD or JTAG ports: 7
3) Configuring the Trace: 8
4) Trace Configuration Fields: 9
5) Program STOP Control: 11
6) Debug Information Display: 11

Part 4: The Rest of the Tracing Examples: 12

1) Creating a Global Variable (to use in our examples): 12
2) Watch window: 13
3) Memory Windows: 13
4) Logic Analyzer: 14
5) Data Reads and Writes in the Trace Records window: 15
6) Watchpoint Example (Access Breaks): 16
7) Breakpoints (Execution Breaks): 17
8) Exceptions: Tracing: 18
9) PC Samples: Tracing: 19
10) Debug (printf) Viewer: 20
11) Opening and using RTX Kernel Awareness: The RTX Viewer: 21
12) Creating a new Task in RTX_Blinky: 23

Part 5: …more Useful Information 24

1) Creating a new µVision project from scratch: 24
2) Running Blinky from RAM: 25
3) SmartFusion eNVM (Flash) Programming: 27
4) TCP/IP: Keil http demonstration: 28
5) ARM JTAG/SWD adapter connector schematics and the New ULINK-ME: 29
6) Acronym List: 30
7) Overloading the SWO pin: 30
8) Top Six Reasons why you can't get SWV working: 31

Copyright © 2010 ARM Ltd. All rights reserved

1) Serial Wire Summary:

We have three basic debug systems as implemented in SmartFusion Cortex-M3 devices:

1. SWV and ITM data output on the SWO pin.
2. DAP Reads and Writes in/out the JTAG/SWD ports.
3. Breakpoints and Watchpoints are set/unset through the JTAG/SWD ports.

These are all completely controlled through µVision via a ULINK.

For more detailed information regarding Serial Wire Viewer please refer to the ULINK*pro* User Manual.

2) Usefulness of the Trace:

SWV Trace adds significant power to debugging efforts. Problems which may take hours, days or even weeks in big projects can often be found in a fraction of these times with a trace. Especially useful is where the bug occurs a long time before the consequences are seen or where the state of the system disappears with a change in scope(s) or RTOS task switches.

Usually, these techniques allow finding bugs without stopping the program. These are the types of problems that can be found with a quality trace:

1) Pointer problems.
2) Illegal instructions and data aborts (such as misaligned writes).
3) Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs)
4) A corrupted stack.
5) Out of bounds data. Uninitialized variables and arrays.
6) Stack overflows. What causes the stack to grow bigger than it should ?
7) Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. ***This is probably the most important use of trace.***
8) Communication protocol and timing issues. System timing problems.
9) Profile analysis and code coverage.

3) What kind of data can the Serial Wire Viewer see ?

1) Global variables.
2) Static variables.
3) Structures.
4) Can see Peripheral registers – just read or write to them. The same is true for memory locations.
5) Can see executed instructions. SWV only samples them.

What Kind of Data the Serial Wire Viewer can't see…

6) Can't see local variables. (just make them global or static).
7) Can't see register to register operations. PC Samples records some of the instructions but not the data values.
8) SWV can't see DMA transfers. This is because by definition these transfers bypass the CPU and SWV can only see CPU actions.

Copyright © 2010 ARM Ltd. All rights reserved

4) Hardware:

We will examine two SmartFusion boards and will focus on the smaller one: the A2F-EVAL-KIT.

A2F-DEV-KIT: See Figure 1. It is connected to a ULINK-ME in the upper right corner. This is referred by Actel as the SmartFusion **Dev**elopment Kit.

A2F-EVAL-KIT: See Figure 2. It is shown with a Keil ULINK2 USB to JTAG/SWD adapter. This is referred by Actel as the SmartFusion **Eval**uation Kit.

Debug Adapters: A Keil ULINK2, ULINK-ME, ULINK*pro* or a Segger J-Link (black case, version 6 or later) can be used. ULINK2 and ULINK-ME are functionally the same. The ULINK*pro* is the same as the other ULINKs but faster flash programming and with the addition of ETM trace support. SmartFusion does not support ETM.

Earlier versions of ULINK-ME provide 3.3 volts power to a target board through JTAG connector pin 1. You might want to remove this by cutting the pin 1 lead or remove the diode on the ME that is connected to pin 1.

You do not need a FPGA design programmed into SmartFusion parts to program and run the Cortex-M3 processor with µVision. You do need the FPGA programmed in order to have the LEDS blink. Get instructions here to do so: www.actel.com/download/rsc/?f=A2F_EVAL_KIT_Mfg_PF for a suitable design.
On page 2, run the section **5.0 Programming the A2F-EVAL-KITBoard (F2 device)**.

Figure 1: DEV-KIT with Keil ULINK-ME

Figure 2: EVAL-KIT with Keil ULINK2

Copyright © 2010 ARM Ltd. All rights reserved

5) Software:

You must use Keil MDK 4.11 or later to get all the required Actel files. The evaluation version is available free from www.keil.com. A license is not needed. You can compile code up to 32K including the Keil examples.

Please install MDK into the default directory of C:\Keil to easier follow the directories in this note. Normally, you install MDK anywhere. You can also install multiple copies and they will not interfere with each other.

Keil provides two examples: Blinky and one using the RTOS: RTX_Blinky. These are found in C:\Keil\ARM\Boards\Actel\SmartFusion.

6) Connecting the Hardware:

A2F-EVAL-KIT: the Small Blue Board: See Figure 3.

1. Connect the debugging adapter to J3 RVI Header, the standard JTAG adapter as shown.
2. JP7 and JP10 jumpers set to position 2-3. They are labeled RVI and M3 respectively.
3. JP6 2-3, J6 installed, J5 and J8 are not installed.
4. Connect a USB cable to power the board to J14. Some LEDs will illuminate.

A2F-DEV-KIT: the Big Green Board: See Figure 4.

1. Connect the ULINK to J3, the standard JTAG adapter as shown in Figure 4.
2. SW9 to S/W Debug ON. (the right most position)
3. JP7 set to 2-3 RVI. (the other position is to program the FPGA Flash)
4. JP6 must be in position 2-3 V1P5_EXT.
5. JP5 both jumpers to C1 – Single Prog. See Figure 4 for a close-up of these jumpers.
6. Power the board with the Actel provided 5 volt AC adapter. If you see a Found New Hardware Wizard message: click Cancel to ignore it as we will not be using the on-board USB port.

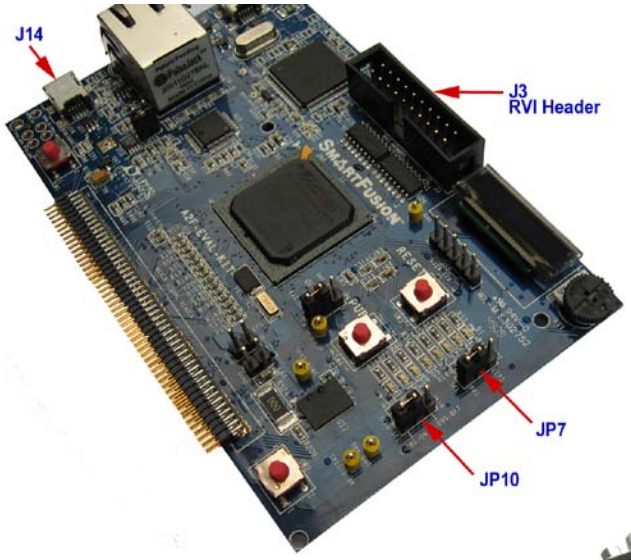


Figure 3: EVAL-KIT Jumper Settings for Keil ULINK operation.

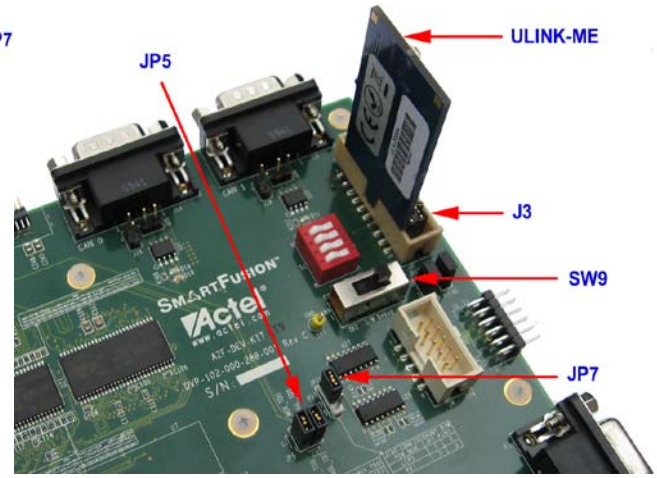


Figure 4: DEV-KIT Jumpers Settings for ULINK operation.

Copyright © 2010 ARM Ltd. All rights reserved

Software Update: MDK 4.11 contains several enhancements. The first are updated CMSIS compliant header files from Actel. The other is a new FLASH programming algorithm to move the example files from the area of 0x6000_0000 to 0x0. This is more consistent with typical Cortex-M3 programming. Programs must be run at less than 0x2000_0000 in order for the hardware breakpoints to work reliably. **You must use MDK 4.11 or later !**

1) *Blinky* example program using the Actel SmartFusion

Now we will run the Keil MDK development system using a SmartFusion evaluation board.

You do not need anything programmed in the FPGA Flash to run Cortex-M3 programs.

1. Connect the hardware as in Figure 1 or 2.
2. Start µVision by clicking on its desktop icon.

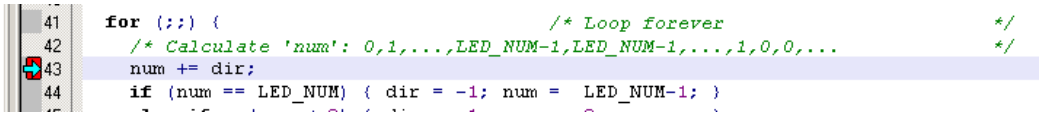
1. Select Project/Open Project from the main menu.
2. Open the file C:\Keil\ARM\Boards\Actel\SmartFusion\Blinky\Blinky.uvproj.
3. Compile the source files by clicking on the Rebuild icon. .
4. Program the SmartFusion eNVM flash by clicking on the Load icon:
5. Enter Debug mode by clicking on the Debug icon. Select OK if the Evaluation Mode notice appears.
6. Click on the RUN icon to start the Blinky program. . ***Blinky is now running !***
7. If the FPGA is programmed to connect the GPIO port to the LEDs, at least one LED will blink.
8. Stop the program with the STOP icon.

Now you know how to compile a program, load it into the SmartFusion Flash, run it and stop it.

LEDs: No LEDs will blink on the EVAL-KIT or the DEV-KIT when the Blinky program is running unless the FPGA is programmed with a hardware design to connect the GPIO port to the LED. This is the one exception where you need some design programmed into the FPGA. See under 4) Hardware on page 4 for a suitable design.

Something Extra to do:

1. Run the program again. .
2. Open the Blinky.c tab. It is probably already open.
3. Scroll down to near Line 43: **`num += dir;`**
4. Note on the left of the line numbers are darker grey blocks. This indicates there is assembly code present and you can set a hardware breakpoint on this line. You can also see these in the Disassembly window.
5. *While the program is still running*, double-click to the left of the 43 and a red block will be created. This is a hardware breakpoint. SmartFusion has 6 but µVision uses one or two and sometimes more.
6. The cyan arrow is where the source and disassembly windows are matching. You set this with your mouse.
7. Note that the program will soon stop at line 43. The yellow arrow is the current program counter position. This will be the next instruction executed when the CPU is started again.



TIP: You can set and unset breakpoints on-the-fly. No need to stop the CPU. This is an important feature.

8. Double-click on the red box to remove the breakpoint.

Copyright © 2010 ARM Ltd. All rights reserved

1) Debug Ports: JTAG, SWD, SWJ and SWO:

JTAG: The JTAG port is used to access the internal CoreSight debug module from the outside world and uses from 4 to 5 pins. A USB to JTAG adapter such as the ULINK2 connects the debugger software to the target chip.

Serial Wire Debug (SWD): SWD is a two wire alternative to the JTAG port. SWD has a bi-directional signal plus a clock. It shares the JTAG signals TMS and TCK respectively on the JTAG connector. SWD can do everything JTAG can except for boundary scan. You must use SWD to use the Serial Wire Viewer.

SWJ: This switches between JTAG and SWD mode. SWJ must be enabled to select either JTAG and SWD.

SWO: Serial Wire Output. This is a single pin high speed output that provides the Serial Wire Viewer trace data. It is easily overloaded with excessive amounts of data. SWO shares a pin with the JTAG signal TDO.

JTAG and SWD: The example above uses the SmartFusion JTAG port to communicate with the ARM CoreSight debugging module. To use Serial Wire Viewer, we will need to use the Serial Wire Debug (SWD) port. SWD can do everything JTAG can except Boundary Scan. See the next page to activate it.

2) Selecting SWD or JTAG:

You must use Single Wire Debug (SWD) mode to use Serial Wire Viewer. This is configured in µVision as shown in Figure 5. You must first have a ULINK connected to a Cortex-M3 target.

1) If necessary, stop the CPU and exit debug mode.
2) Click on the Target Options icon or select "Project/Options for Target" or pressing Alt+F7.
3) Click on the Debug tab.
4) Click Settings beside the USE: ULINK Cortex Debugger box to open Figure 5.

The box labeled SW Device (or labeled JTAG Device Chain if JTAG is selected) in Figure 5 will display the CoreSight devices detected in the processor or an error if none are found.

5) Select SWJ and SW in the Port: box. Confirm SW-DP is shown as in Figure 5. You must see SW-DP before you continue to Step 6.

Figure 5 shows that the ULINK is now successfully been connected to the CoreSight SWD port.

Urgent TIP: If you don't see either SW-DP or JTAG-DP in this window and get an error message or device not found – stop right now and fix this. This section ***must*** work before you can continue any debugging. **Make sure the jumpers are set correctly** and the board is properly powered. See **TRAP** below. Try re-powering the board.

TRAP: Top reason people can't get SW and/or JTAG working is a wrong setting of the jumpers JP7 and JP9 (on EVAL board) and the two JP5 jumpers on the larger DEV board. This is because these must be switched to enable FPGA programming. By default they are set to FPGA programming and not for standard SWD or JTAG operation.

6) Click on OK twice to return.

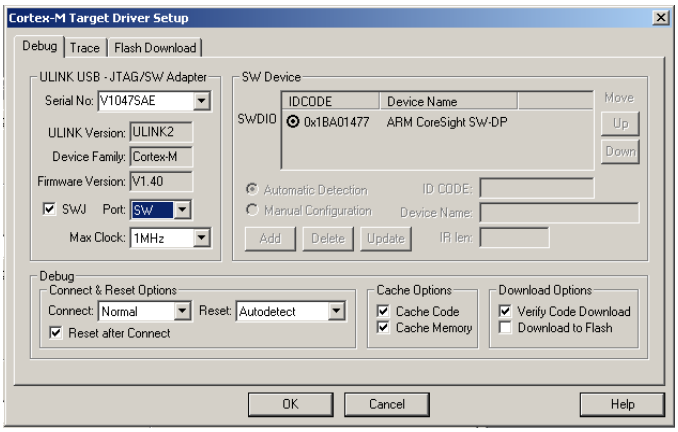


Figure 5: Selecting SWD Mode

TIP: To refresh this window switch between JTAG and SW in the Port: drop down menu.

Copyright © 2010 ARM Ltd. All rights reserved

3) Configuring the Trace:

In order to see the Serial Wire Viewer features, the Trace must be configured as in Figure 6.

1) If necessary, stop the CPU and exit debug mode.
2) Click on the Target Options icon or select “Project/Options for Target”.
3) Click on the Debug tab.
4) Click Settings beside the USE: ULINK Cortex Debugger box. Confirm SWJ and SW are selected.
5) Click on the Trace tab to open Figure 6.
6) Set Core Clock to 0.098 MHz. Note: this will change in the next SmartFusion silicon release.
7) Select Trace Enable and EXCTRC.
8) Set the Timestamps Prescaler to 1 to lessen data overruns.
9) Unselect Periodic and on Data R/W Sample. Window will look like Figure 6.
10) Click on OK twice to return to the main screen.
11) The Serial Wire Viewer is now activated.
12) Click on File/Save All to save these settings.

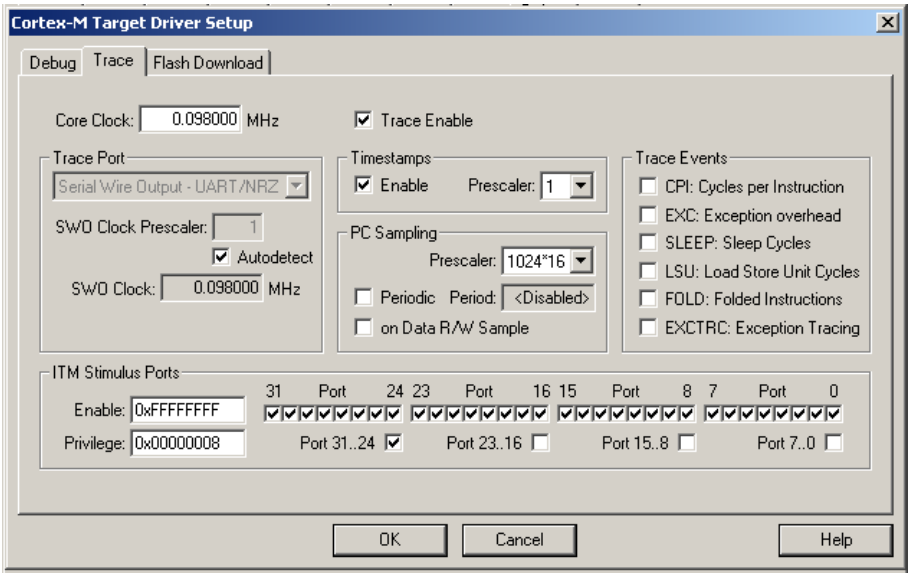


Figure 6: Trace Configuration window.

TIP: How to open this window again to modify trace selections:

1) **In Edit mode:** The Target Options icon as already described.
2) **In Debug Mode:** Debug/Debug Settings from the main menu.

You select various SWV elements in Figure 6. It is important to note that it is easy to overload the single wire Serial Wire Output (SWO) pin. The general rule is to activate only those features you need.

TIP: You can only enable/disable trace from Edit mode. You can change the rest while in either mode.

The next page describes in detail the various fields of the Trace Configuration window.

Two pages after that describes some of the attributes of Serial Wire Viewer and the Debug Access Port (DAP) data read and write operations.

Copyright © 2010 ARM Ltd. All rights reserved

4) Trace Configuration Fields:

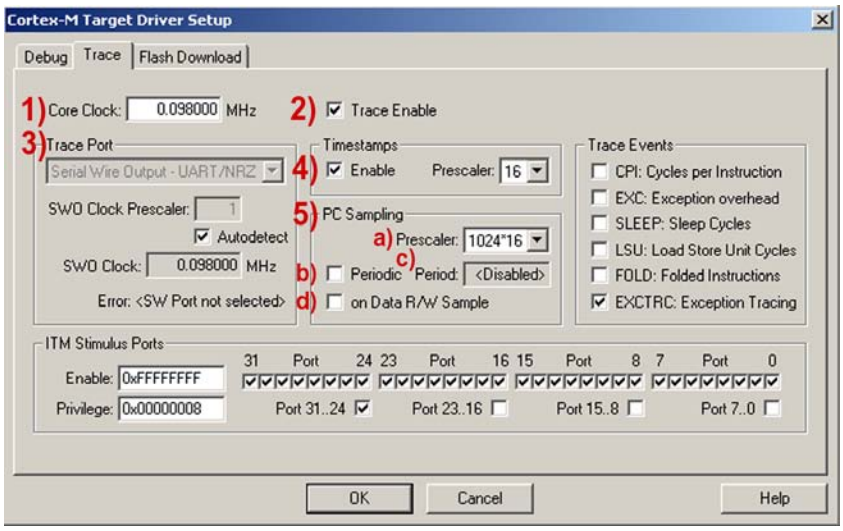


Figure 7: Trace Configuration window Fields 1) through 5).

1) **Core Clock:** The CPU clock speed for SWV. SWO Clock signal is derived from and is a ratio of the Core Clock. Current ratio for SmartFusion is CPU/1024. 100 MHz/1024 = 98 KHz. This value will change in the next silicon release. Core Clock will then equal 100 MHz.
2) **Trace Enable:** Enables SWV and ITM which is essentially everything on this window except Trace Port. This does not affect the DAP Watch and Memory window display updates.
3) **Trace Port:** Selects the SWO trace output UART or Manchester protocol.
    a. **Serial Wire Output – UART/NRZ:** ULINK2 and ULINLK-ME must use this. UART/NRZ encoding is not supported by ULINK*pro*.
    b. **Serial Wire Output – Manchester**: Use Manchester encoding. ULINK*pro* only.
4) **Timestamps:** Enables timestamps and selects a Prescaler. 1 is the default. Selecting a higher value can, but not always lessen SWO overloads. Completely disabling the timestamps can lessen data overruns but can disable other SWV features. It is worth a try if you are having overload problems.
5) **PC Sampling:** Samples the program counter and displays in the Trace Records window.
    a. **Prescaler** 1024*16 (the default) means every 16,384th PC is displayed. The rest are lost.
    b. **Periodic:** Enables PC Sampling.
    c. **Period:** Automatically derived from Prescaler and Core Clock settings.
    d. **On Data R/W Sample:** Displays the address of the instruction that made a data read or write of a variable entered in the Logic Analyzer in the Trace Records window. This is not connected with PC Sampling.

TIP: It is important to ensure the Serial Wire Output (SWO) pin is not overloaded. µVision will alert you when an overflow occurs with a "X" in the Trace Records window or with a "D" or a "O" in the ULINK*pro* Instruction Trace window. µVision easily recovers from these overflows and immediately continues displaying the next available trace frame. Dropped frames are somewhat the normal situation especially with many data reads and/or writes..

TIP: ULINK*pro* can process SWV information much faster than the ULINK2 or ULINK-ME can. This results in fewer dropped frames especially with high data transfer rates out the SWO pin. Data overruns are often associated with a fast stream of data reads and writes which are set in the Logic Analyzer. Minimize these issues by displaying only the information you really need.

Copyright © 2010 ARM Ltd. All rights reserved

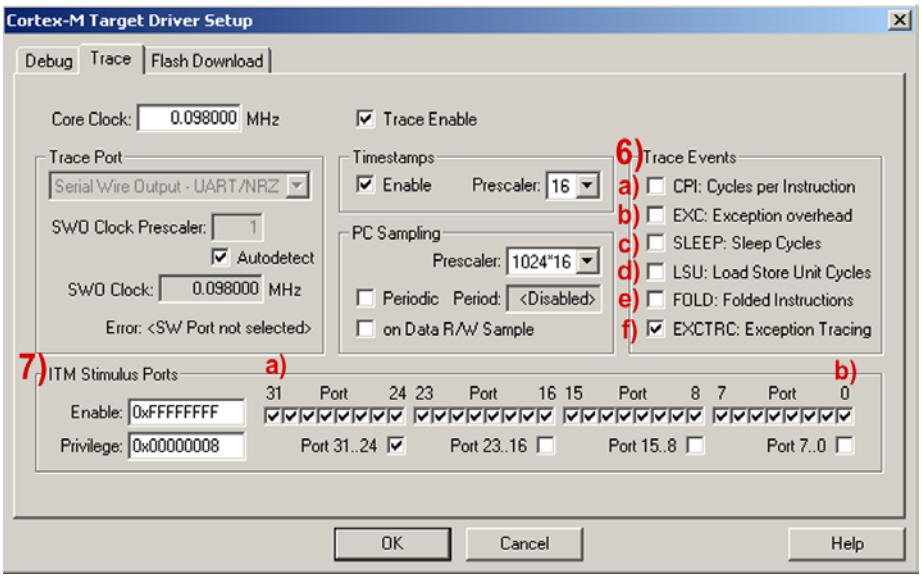


Figure 8: Trace Configuration window Fields 6) through 8)

6) Trace Events: Enables various CPU counters. All except EXCTRC are 8 bit counters. Each counter is cumulative and an event is created when this counter overflows every 256 cycles. These values are displayed in the Counter window. The event created when a counter wraps around is displayed in the Trace Records window or the Instruction Trace window (ULINK*pro*).
Event Counters are updated using the DAP and not SWV. These events are memory mapped and can be read by your program or the µVision Memory window.

- **a. CPI: Cycles per Instruction:** The cumulative number of extra cycles used by each instruction beyond the first one plus any instruction fetch stalls.
- **b. Fold:** Cumulative number of folded instructions**.** This will result from a predicted branch instruction removed and flushed from the pipeline giving a zero cycle execution time.
- **c. Sleep:** Cumulative number of cycles the CPU is in sleep mode. Uses FCLK for timing.
- **d. EXC:** Cumulative cycles CPU spent in exception overhead not including total time spent processing the exception code. Includes stack operations and returns.
- **e. LSU:** Cumulative number of cycles spent in load/store operations beyond the first cycle.
- **f. EXCTRC:** Exception Trace. This is different than the other items in this section. This enables the display of exceptions in the Instruction Trace and Exception windows. It is not a counter. This is a very useful feature and is often used in debugging.

7) ITM Stimulus Ports: Enables the thirty two 32 bit registers used to output data in a *printf* type statement to µVision. Port 0 is used for the Debug (printf) Viewer, Port 31 is used for the Keil RTX real-time kernel awareness window. Only Ports 0 and 31 are currently implemented in µVision and should normally be checked. Ports 1 through 30 are not currently implemented and are Don't Care.

- **a. Port 31:** Enables the ITM port used for the RTX Viewer.
- **b. Port 0:** Enables the ITM port used for the Debug (printf) Viewer. A small amount of instrumentation code is needed in your project. See 10) Debug (printf) Viewer in Part 4 for information on using this feature.

Copyright © 2010 ARM Ltd. All rights reserved

5) Program STOP Control:

Breakpoints: SmartFusion has six hardware breakpoints. The µVision debugger will use one, perhaps two for its operation. µVision will alert you if you attempt to use more breakpoints than are available. Breakpoints can be set on-the-fly while the program is running as long as the program is located at addresses 0x2000 0000 or lower. For addresses greater than this, you must stop the program to set a hardware breakpoint.

Watchpoints: Sometimes referred to as Access Breaks in Keil documents. These can be thought of conditional breakpoints. Two are available in SmartFusion processors. They can't be set on the fly.

TIP: See Watchpoints and Breakpoints in Part 4.

TIP: Breakpoints and Watchpoints are different – they originate in different parts of CoreSight. For instance, killing all breakpoints inside µVision does not kill the Watchpoints.

6) Debug Information Display:

When we speak of "trace" we can refer to Serial Wire Viewer (SWV) or ETM. Both produce historical records concerning the execution of the program.

a) Serial Wire Viewer (SWV): SWV is a term used to describe some debugging features of CoreSight. Specifically, this refers to the real-time display of PC Samples, Exceptions (including interrupts), data reads and writes, ITM, CPU counters and a timestamp. These features will be described presently. Most of the SWV information comes out the SWO pin. The "Read and Write to Memory Locations" described below can also be thought of as part of SWV. However, this data comes out of the processor via the SWD or JTAG DAP port rather than the SWO pin. The Event Counters also come out the DAP.

b) Read and Write to Memory Locations: µVision has the ability to read and write memory locations on-the-fly and without stealing CPU cycles by using the Debug Access Port (DAP). The Cortex-M3 is a Harvard architecture: this means it has separate code and data address busses. While the CPU is fetching instructions at full speed from the code space, µVision can access the data space via JTAG or SWD. This feature is active even while the CPU is halted.

This feature is used to display data in the Watch and Memory windows as well as the RTX awareness window. Not only can values of memory locations be displayed and updated in real-time: you can also insert values while the program is running. The only instance real-time will be violated is in the unlikely event the CPU accesses a memory location the exact same time as µVision does.

Copyright © 2010 ARM Ltd. All rights reserved

1) Creating a Global Variable:

We will add a Global Variable to use in our demonstrations:

1. Open the source file Blinky.c by clicking on its tab.
2. Create a global variable **`count`** near line 18 as in Figure 9.
   `int count = 0;`
3. Add the line **`count++;`** near line 48.
4. If in Debug mode, exit it.
5. Compile the project. You must get no errors or warnings. See the bottom left corner for the Build Output window.
6. Program the Flash.
7. Enter Debug mode.
8. Click on RUN.
9. Leave this running for the Watchpoint example on the next page.

```
16  const unsigned long led_mask[] = { 1 <<
17  volatile unsigned long SysTickCnt;
18  int count=0;
19  void SysTick_Handler (void) {
20    SysTickCnt++;
21  }
22
23  void Delay (unsigned long tick) {
24    unsigned long systickcnt;
25
26    systickcnt = SysTickCnt;
27    while ((SysTickCnt - systickcnt) < t:
28  }
29
30
31  int main (void) {
32    int num = -1;
33    int dir =  1;
34
35    WATCHDOG->WDOGENABLE = 0x4C6E55FA;
36
37    SysTick_Config(SystemCoreClock/1000);
38
39    GPIO->GPIO_6_CFG = 5;
40
41    for (;;) {
42      /* Calculate 'num': 0,1,...,LED_NUI
43      num += dir;
44      if (num == LED_NUM) { dir = -1; nur
45      else if    (num < 0) { dir =  1; nur
46      GPIO->GPIO_OUT &= ~led_mask[num];
47      Delay(500);
48      count++;
49      GPIO->GPIO_OUT |=  led_mask[num];
50      Delay(500);
51    }
```

Figure 9: Variable `count`.

TIP: The changes you made can be saved in the File menu by selecting Save All. It will also be saved if you close the source file, change to a different project or shut µVision down.

TIP: You can edit a source file while in Debug or Edit mode. You must be in Edit mode to compile it.

TIP: The Serial Wire Viewer is unable to display local variables and DMA transfers. It can see static, volatile and global variables, structures and peripheral read and write operations. See the Appendix for details.

TIP: To see a local variable simply make it static: `static int variable_name;`

Copyright © 2010 ARM Ltd. All rights reserved

2) Watch Windows: Updated in Real-time !

Blinky should still be running from the adding of the global variable count on the previous page.

1. In the Watch window, double-click or press F2 and enter **`count`**. Press Enter.
2. You can do this while the program is running. The CPU is not slowed down or stopped.
3. You can also Drag 'n Drop a variable into various windows.
4. The value of count will be displayed and updated in real-time without stealing any CPU cycles.
5. Double-click on the value of count and enter 0 and press Enter. See Figure 10.
6. 0 will be inserted into the memory where count is stored and will continue incrementing.

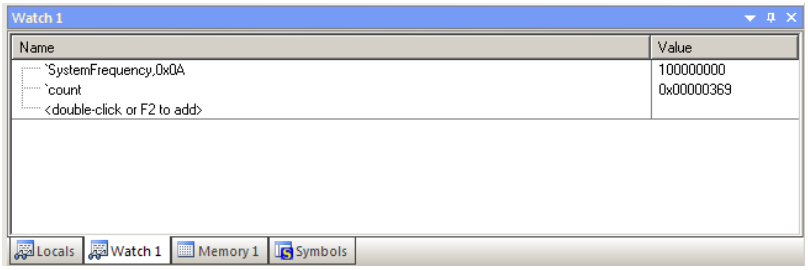


Figure 10: Watch Window

TIP: To Drag 'n Drop into a tab that is not active such as the Logic Analyzer, Watch or Memory windows, pick up the variable by blocking it, click and hold, and move it over the tab you want to open; when the tab opens, move your mouse into the window and release the variable.

3) Memory Windows: also are updated in Real-time !

1. Click on the Memory tab to open it.
2. Enter count in the space provided. The program can be running. See Figure 11.
3. Note that as count increments, the address it points to changes. This is useful for work with pointers.
4. Add an ampersand (&) in front of count. i.e. **`&count`**
5. Note the memory address changes to 0x2000 0010 in this case.
6. This is the physical address where the variable **`count`** is stored.
7. Note you can right click to vary the format the data is displayed. You can also insert a new value.

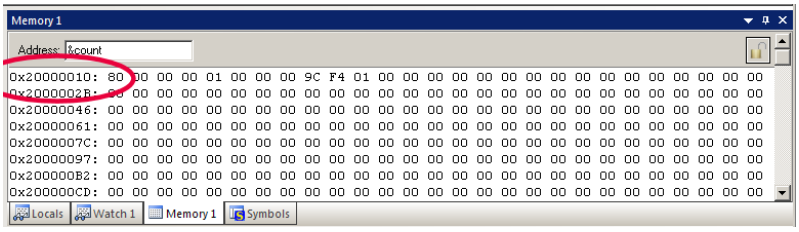


Figure 11: Memory Window

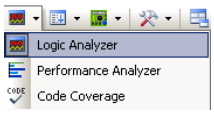
How It Works: Read and Write to Memory Locations: µVision has the ability to read and write memory locations on-the-fly and without stealing CPU cycles by using the Debug Access Port (DAP). The Cortex-M3 is a Harvard architecture: this means it has separate code and data address busses. While the CPU is fetching instructions at full speed from the code space, µVision can access the data space via JTAG or SWD. This feature is active even while the CPU is halted. SWV is not used. It can be disabled.

This feature is used to display data in the Watch and Memory windows, Event Counters as well as the RTX Awareness window. Not only can values of memory locations be displayed and updated in real-time: you can also insert values while the program is running. The only instance real-time will be violated is in the unlikely event the CPU accesses a memory location the exact same time as µVision does.

Copyright © 2010 ARM Ltd. All rights reserved

4) Logic Analyzer Window: Updated in Real-time !

The Logic Analyzer (LA) displays up to four variables in a graphical format. When a variable is placed in the LA, it is also displayed in the Trace Records window.



1. µVision must be in Debug mode. The program can be running.
2. Open View/Analyzer Windows/Logic Analyzer or by the drop down menu if the LA is not already displayed.
3. Select the LA tab and select Setup in the LA window. Figure 12 will open up.
4. Click on the Insert icon in Figure 12, or press the PC keyboard INSERT key.
5. Enter our global variable **`count`**. You can also drag and drop the variable into the LA window.
6. Select the Display Range max = 0xFF and Min = 0. Click on Close.
7. If you do not see **`count`** in the window – stop and start the program.
8. Click on RUN if not already running. LA window will be updating as in Figure 13.
9. If necessary, click on Zoom: Out for an appropriate X axis. Try 10,000 seconds.
10. If the Logic Analyzer doesn't update: try clicking on the scroll bar.
11. Add **`count`** to the Watch window if it is not already there.
12. In the Watch window: change the value of count to various values to change LA values.

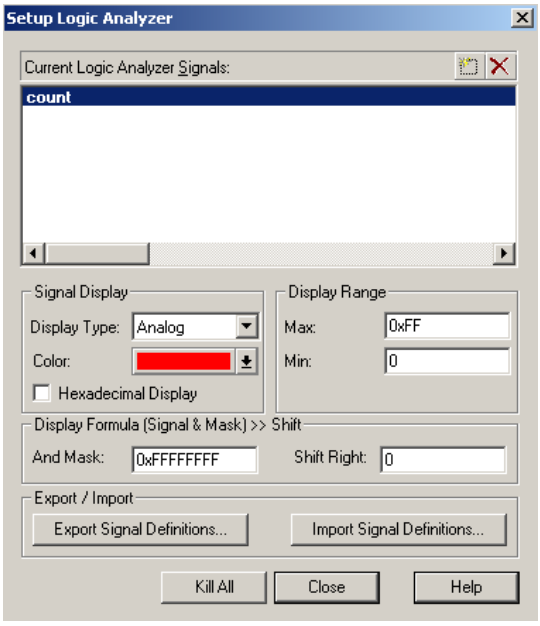


Figure 12: Logic Analyzer Setup window.

Figure 13 Explanation:

A: count is incrementing. Note Range = 100K sec.

B: count is manually set to zero in Watch window.

C: count is manually set to 0xF0 in the Watch window.

D: count is manually set to zero in the Watch window.

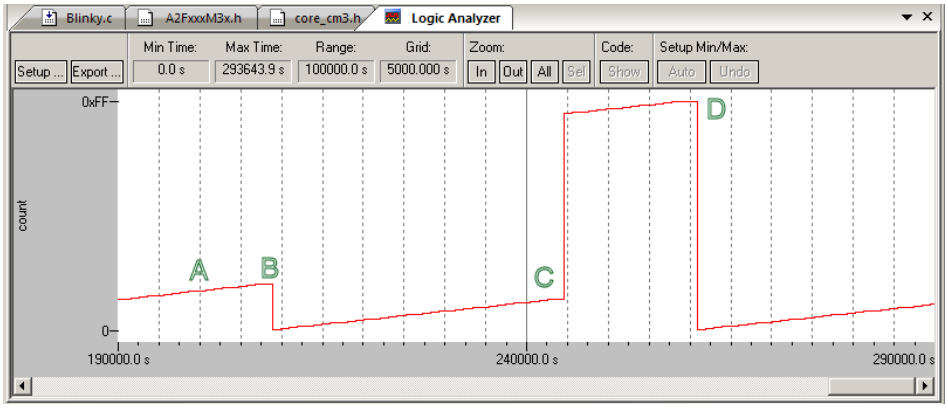


Figure 13: Logic Analyzer windows displaying global variable count.

Crucial TIP: If the Logic Analyzer data does not increment but the variable does in the Watch or Memory windows, this can mean the SmartFusion SWO is overloaded. Sometimes the LA will update only by clicking on the scroll bar or stopping the program. Disable some of the SWV functions such as exceptions. Disabling Timestamps often stops the LA. **Try reducing the Range: to 10 seconds or so as a high range can stop the LA update process.**

Copyright © 2010 ARM Ltd. All rights reserved

Logic Analyzer Notes:

1. The Logic Analyzer displays up to four variables in a graphical format.
2. SWV is used to obtain the data displayed. This comes out the Serial Wire Output pin.
3. You will not be able to enter variables if the SWV Trace configuration is not enabled and configured properly. The Core Clock needs to be accurately set.
4. Each variable will have its read and write operations displayed in the Trace Records window.
5. You must enter a variable name. Raw memory addresses are allowed. See the TIP: below.
6. You might have to fully qualify a variable. An example is \Blinky\AD_dbg.
7. Number 1 reason can't enter a variable is Trace not properly set. Test SWV using PC Samples.
8. You can drag and drop variables into the LA. Variables must be global, static or a structure.
9. You may need to modify the range to greater than 5 seconds to clearly display most variables.
10. The LA is updated at a fixed rate of approximately 500 msec.

TIP: Raw addresses can also be entered into the Logic Analyzer.

An example is: ***((unsigned long *)0x20000000)**

5) Data Reads and Writes in the Trace Records for variable count:

When a variable is entered into the Logic Analyzer, the associated data reads and/or writes are displayed in the Trace Records window. This is the method to enter reads and writes in the Trace Records window.

1. Run the program.
2. Open the Trace Records window with View/Trace/Records or the pull-down menu:

3. Figure 14 opens up. Data Reads and Writes to variable count (at 0x2000_0010) are displayed.
4. Note the data values are incrementing. Double-click in the Trace Records window to clear it.
5. Open Debug/Debug Settings and select the Trace tab.
6. Select on Data R/W sample and click on OK twice. This will add the PC column.
7. Click on RUN.

Figure 14 Explanation:

1) Data Read to `count` address 0x2000_0010 the data value 00 by the instruction at 0x6000 03E8.

2) Data Write to `count` address 0x2000_0010 the data value 01 by the instruction at 0x6000 03EE.

The time these events occurred is timed by both CPU cycles and in seconds (Time(s)).

The X in the DLY column indicates the timestamp is delayed from its true value because of overflow.

Trace Records

Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
Data Read			20000010H	00000000H	600003E8H	X	5975073179	60970.13447959
Data Write			20000010H	00000001H	600003EEH	X	5975207031	60971.50031633
Data Read			20000010H	00000001H	600003E8H	X	6075072180	61990.53244898
Data Write			20000010H	00000002H	600003EEH	X	6075205233	61991.89013265
Data Read			20000010H	00000002H	600003E8H	X	6175071179	63010.93039796
Data Write			20000010H	00000003H	600003EEH	X	6175204195	63012.28770408
Data Read			20000010H	00000003H	600003E8H	X	6275070180	64031.32836735
Data Write			20000010H	00000004H	600003EEH	X	6275203217	64032.68588776
Data Read			20000010H	00000004H	600003E8H	X	6375069180	65051.72632653
Data Write			20000010H	00000005H	600003EEH	X	6375202575	65053.08750000
Data Read			20000010H	00000005H	600003E8H	X	6475068178	66072.12426531
Data Write			20000010H	00000006H	600003EEH	X	6475201280	66073.48244898
Data Read			20000010H	00000006H	600003E8H		6575067181	67092.52225510
Data Write			20000010H	00000007H	600003EEH	X	6575200618	67093.88385714
Data Read			20000010H	00000007H	600003E8H	X	6675066178	68112.92018367
Data Write			20000010H	00000008H	600003EEH	X	6675199637	68114.28201020

Figure 14: Trace Records displaying read and write operations to variable count.

Copyright © 2010 ARM Ltd. All rights reserved

6) Watchpoints: (also known as Access Points)

This is an excellent opportunity to show how the Watchpoints work in Keil µVision.

Watchpoints are also known as Access Breaks and are indicated by an **(A** in Figure 16. They are set in the Breakpoints Window. Watchpoints can be thought of as conditional breakpoints when compared to an Execution breakpoint. Execution breakpoints stop the CPU when a specified instruction is fetched (but not executed). Watchpoints stop the CPU when a specified data access occurs and any specified expression becomes true. There might be some skid. This is useful for locating read and write operations you do not expect.

SmartFusion has 2 Watchpoints. Watchpoints must be configured in Debug mode and the CPU is halted.

How to set a Watchpoint:

1. Stop the program and remain in Debug mode.
2. Open the Breakpoint window through View/Breakpoints or Ctrl-B. Figure 15 opens up.
3. Watchpoints are created in the bottom half of Figure 15 and defined and listed in the top half.
4. Enter the count in the bottom half of the Breakpoints window: Figure 15. Click Read and Write.
5. Figure 15 says the program will be halted if the value of 0x22 is written to or read ***once*** from the variable count. (Count = 1) The other entries are the default settings.
6. Click on Define and the Watchpoint moves to the upper half as in Figure 16. Click on Close.
7. Set count < 0x22 in the Watch window if necessary or the Watchpoint will never activate.
8. The Trace Records window must be open. Double-click on it to clear it.
9. Click on RUN and wait for the program to stop.
10. Scroll to the bottom of the Trace Records window and the data write is displayed as in Figure 17.

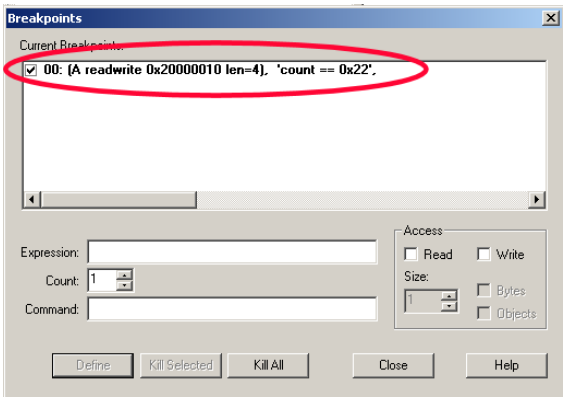


Figure 15: Setting up count

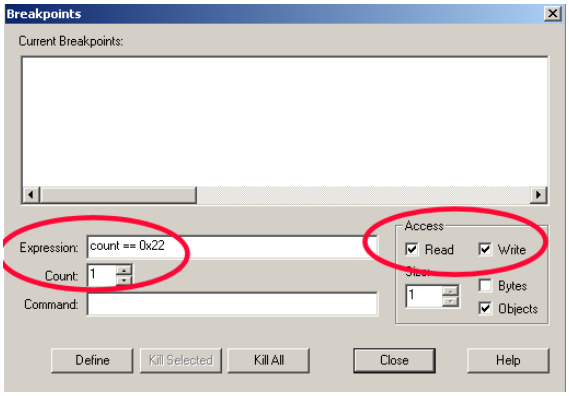


Figure 16: count Defined

TIP: In the definition of a Watchpoint, legal operatives are &, &&, <, >, <=, >=, == and !=.

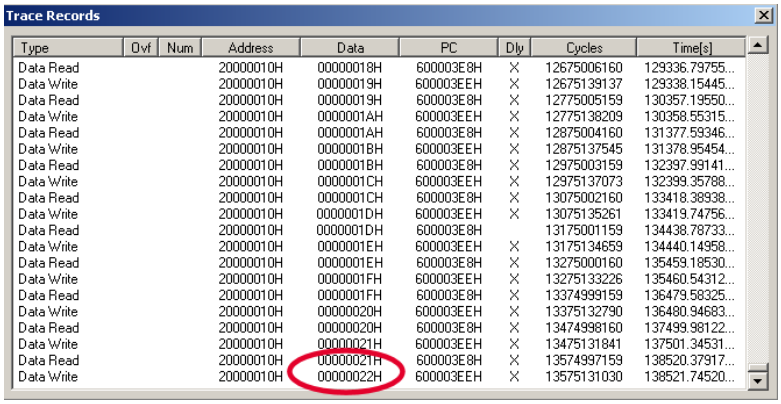


Figure 17: Trace Records with data write of 0x22 to variable count.

Copyright © 2010 ARM Ltd. All rights reserved

7) Breakpoints: (Execution Breakpoints)

These breakpoints can be set on the fly while the program is running: just double-clink in the left margin of a source window. When the PC points to this instruction, the CPU will be stopped before it is executed.

These are called Execution Breakpoints indicated by the (E) in Figure 19. They are set on instructions in the code space. Never on Data. See Watchpoints for operations on data accesses.

Setting Breakpoints:

There are two ways to set a breakpoint (BP):

1. Double-click on the left margin in a source or disassembly file: even while the program is running. Figure 18 shows the set breakpoint by the red box. This must be set on a gray box which indicates there is an assembly instruction present. Line 45 has no assembly instructions while 46 does.

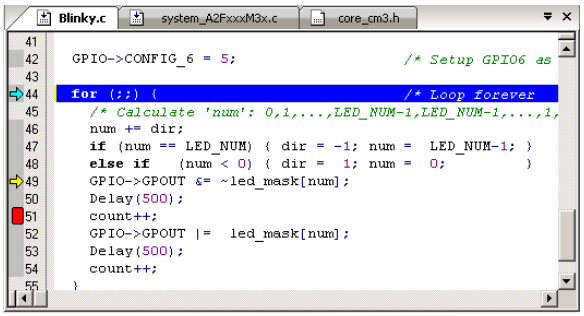


Figure 18: Breakpoint set on Line 51

2. The Breakpoints window. See Figure 19. Open it with Debug/Breakpoints… or Ctrl-B. It is accessible only in Debug mode and when the program is stopped. Watchpoints (Access Breaks) are also entered in the Breakpoints window.

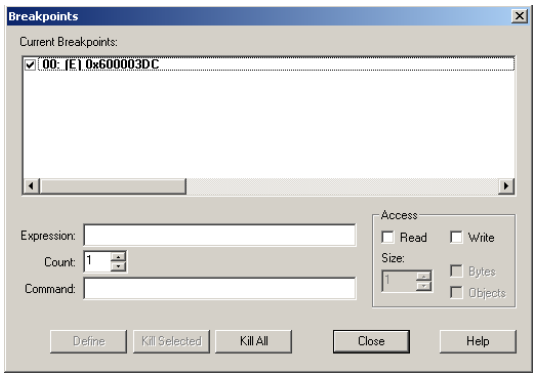


Figure 19: Breakpoints window

To create an entry:

1. Enter the appropriate data in the window.
2. Click on Define.
3. Click on Close.

To modify an entry:

1. Double-click on the entry. It will be brought down into the bottom area.
2. Make your modifications.
3. Click on Define to enter it.
4. You will have to delete the original or disable it by un-checking the box.

Notes:

1. Hardware BPs are not eligible for all options shown in Figure 19 although Count is available.
2. You can disable a BP by un-checking the box in Figure 19 in the Current Breakpoints area.
3. µVision will allow only the available BPs to be set.

TIP: This is the same window used to create and modify Watchpoints.

TIP: If an expression points to code space: an execution or hardware breakpoint is created.

TIP: If an expression points to a data address, a Watchpoint or Access Break is created.

TIP: If you attempt to enter more Breakpoints or Watchpoints than the device has: µVision will warn you.

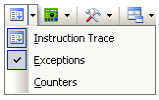
Copyright © 2010 ARM Ltd. All rights reserved

8) Exception Tracing:

Serial Wire Viewer displays exceptions and interrupts in real-time without stealing any CPU cycles or needing instrumentation code. Note that ARM interrupts are a subset of exceptions.

The Keil examples have the SysTick timer running. We will use this exception as an example.

1. Open the Trace Configuration window by clicking on Debug/Debug Settings.
2. Enable EXCTRC: Exception Tracing and click on OK twice to return.
3. Click on RUN to start the program.
4. Open the Exception Trace window and ensure the Trace Record window is still open.
5. The Exception Trace window will display the SysTick timer in Figure 18. Double-click to reset.
6. Scroll up and down and you can see the listing of all potential exceptions available in the NVIC.



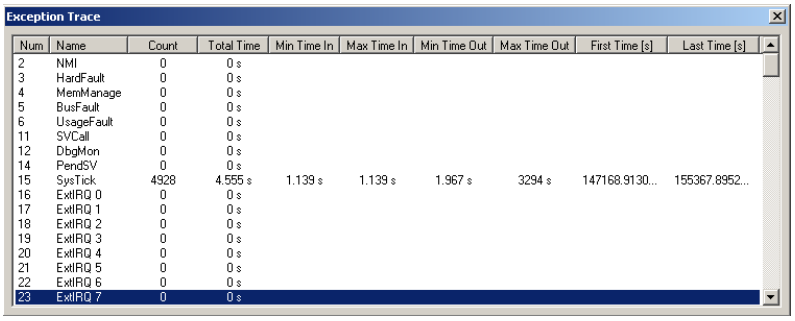

Exception Trace
Num Name Count Total Time Min Time In Max Time In Min Time Out Max Time Out First Time [s] Last Time [s]
2 NMI 0 0 s
3 HardFault 0 0 s
4 MemManage 0 0 s
5 BusFault 0 0 s
6 UsageFault 0 0 s
11 SVCall 0 0 s
12 DbgMon 0 0 s
14 PendSV 0 0 s
15 SysTick 4928 4.555 s 1.139 s 1.139 s 1.967 s 3294 s 147168.9130... 155367.8952...
16 ExtIRQ 0 0 0 s
17 ExtIRQ 1 0 0 s
18 ExtIRQ 2 0 0 s
19 ExtIRQ 3 0 0 s
20 ExtIRQ 4 0 0 s
21 ExtIRQ 5 0 0 s
22 ExtIRQ 6 0 0 s
23 ExtIRQ 7 0 0 s

Figure 20: SysTick Timer.

1. Right click on the Trace Records window and filter out Data Read and Write frames. Exceptions will be left as in Figure 121.
2. Note the X in the OVF column. Some frames were lost due to overloading the SWO pin.
3. Remove **count** from LA, disable On R/W samples and TimeStamp in the Trace Configuration window. Return and run the program and overflows will be gone. But so will the timestamps.

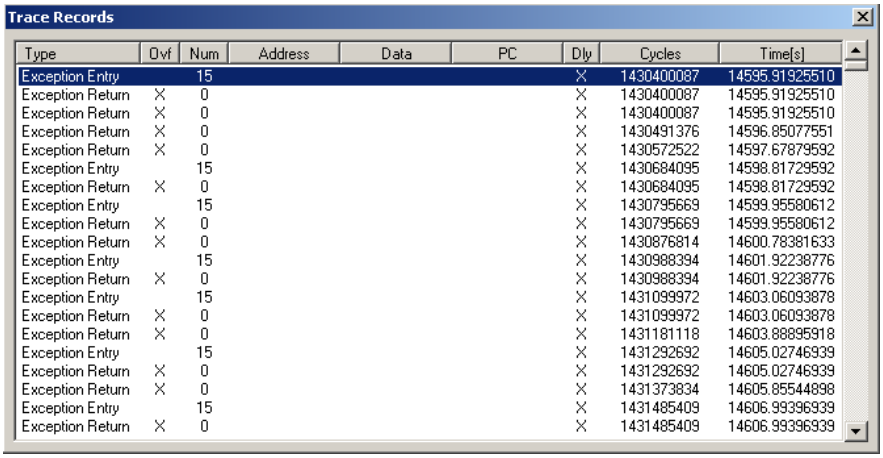


Figure 21: Exceptions in the Trace Records window.

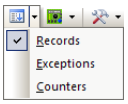
Explanation of Exception frames:

- **Entry:** when the exception enters.
- **Exit:** When it exits or returns.
- **Return:** When all the exceptions have returned including any Cortex-M3 tail-chaining.

Copyright © 2010 ARM Ltd. All rights reserved

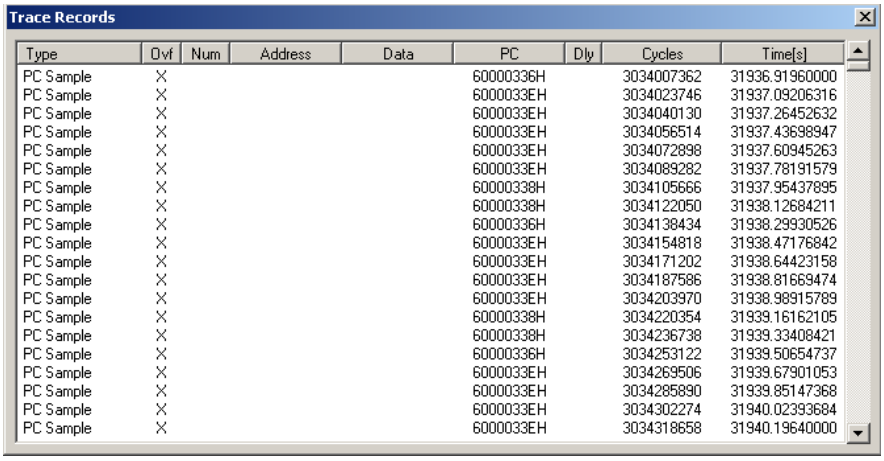
9) PC Samples Tracing:

µVision can display a sampling of the program counter values in the Trace Records window. PC Samples is configured in the Trace Records window. This window is opened with View/Trace/Records or the drop down menu shown here:



1. Open the Trace Configuration window by clicking on Debug/Debug Settings.
2. Enable PC Samples by checking Periodic in the PC Sampling area.
3. Disable EXCTRC: and the TimeStamps.
4. Click on OK twice to return.
5. Click on RUN to start the program.
6. Open the Trace Records window. PC Samples will be displayed as shown in Figure 22.

The values displayed in the PC column gives you an indication where the program is spending its time.

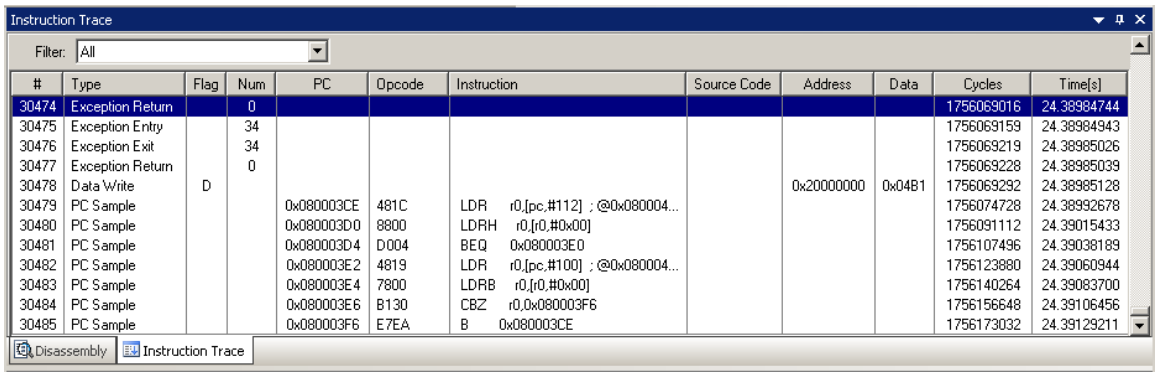


Type	Ovf	Num	Address	Data	PC	Dly	Cycles	Time[s]
PC Sample	X				60000336H		3034007362	31936.91960000
PC Sample	X				6000033EH		3034023746	31937.09206316
PC Sample	X				6000033EH		3034040130	31937.26452632
PC Sample	X				6000033EH		3034056514	31937.43698947
PC Sample	X				6000033EH		3034072898	31937.60945263
PC Sample	X				6000033EH		3034089282	31937.78191579
PC Sample	X				60000338H		3034105666	31937.95437895
PC Sample	X				60000338H		3034122050	31938.12684211
PC Sample	X				60000336H		3034138434	31938.29930526
PC Sample	X				6000033EH		3034154818	31938.47176842
PC Sample	X				6000033EH		3034171202	31938.64423158
PC Sample	X				6000033EH		3034187586	31938.81669474
PC Sample	X				6000033EH		3034203970	31938.98915789
PC Sample	X				60000338H		3034220354	31939.16162105
PC Sample	X				60000338H		3034236738	31939.33408421
PC Sample	X				60000336H		3034253122	31939.50654737
PC Sample	X				6000033EH		3034269506	31939.67901053
PC Sample	X				6000033EH		3034285890	31939.85147368
PC Sample	X				6000033EH		3034302274	31940.02393684
PC Sample	X				6000033EH		3034318658	31940.19640000

Figure 22: PC Samples from the ULINK2 or ULINK-ME

TIP: PC Samples are very liable to overload the SWO pin because of the relatively low 98 KHz speed of the current SmartFusion silicon. This will be increased in the next rev of the SmartFusion silicon. All other functions, including the Logic Analyzer, must be turned off in order for the PC Samples to update. Otherwise, the program must be stopped to display the PC Samples. Note the X in the Ovf column indicates many PC Sample frames were lost. This is still useful information.

ULINK*pro* Instruction Trace Window: The ULINK*pro* provides the Instruction Trace window shown in Figure 23. Figure 21 shows two Exceptions, a Data Write and seven PC Samples. Note disassembled instructions are shown and if available, source code will also be displayed. If you double-click on a PC Sample you will be taken to that instruction in the disassembly and/or source window. The ULINK*pro* can process SWV data very fast and therefore is much less susceptible to data overruns.



#	Type	Flag	Num	PC	Opcode	Instruction	Source Code	Address	Data	Cycles	Time[s]
30474	Exception Return		0							1756069016	24.38984744
30475	Exception Entry		34							1756069159	24.38984943
30476	Exception Exit		34							1756069219	24.38985026
30477	Exception Return		0							1756069228	24.38985039
30478	Data Write	D						0x20000000	0x04B1	1756069292	24.38985128
30479	PC Sample			0x080003CE	481C	LDR r0,[pc,#112] ; @0x080004...				1756074728	24.38992678
30480	PC Sample			0x080003D0	8800	LDRH r0,[r0,#0x00]				1756091112	24.39015433
30481	PC Sample			0x080003D4	D004	BEQ 0x080003E0				1756107496	24.39038189
30482	PC Sample			0x080003E2	4819	LDR r0,[pc,#100] ; @0x080004...				1756123880	24.39060944
30483	PC Sample			0x080003E4	7800	LDRB r0,[r0,#0x00]				1756140264	24.39083700
30484	PC Sample			0x080003E6	B130	CBZ r0,0x080003F6				1756156648	24.39106456
30485	PC Sample			0x080003F6	E7EA	B 0x080003CE				1756173032	24.39129211

Figure 23: Instruction Trace window from ULINK*pro*.

Copyright © 2010 ARM Ltd. All rights reserved

10) Debug (printf) Viewer:

ITM Stimulus Port 0 is available for a *printf* type of instrumentation that requires minimal user code. After the write to the ITM port, zero CPU cycles are required to get the data out of the processor and into the µVision Debug (printf) Viewer for display. You write ASCII values to Port 0.

1. Stop the program and exit debug mode if necessary.
2. Add this code to Blinky.c. A good place is right after the place where you declared count variable: (the following code is a pointer to ITM Port 0)

```
#define ITM_Port8(n)   (*((volatile unsigned char *)(0xE0000000+4*n)))
```

3. Near line 51 enter these three lines: Just after the line `count++;` that you entered is a good place.

```
ITM_Port8(0) = 0x35;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0D;
while (ITM_Port8(0) == 0);
ITM_Port8(0) = 0x0A;
```

4. Rebuild the source files, program the Flash memory and enter Debug mode.
5. Open Debug/Debug Settings and select the Trace tab. Confirm ITM Port 0 is selected, unselect Periodic, EXCTRC, on Data R/W Sample and TimeStamps Enable as shown in Figure 24.

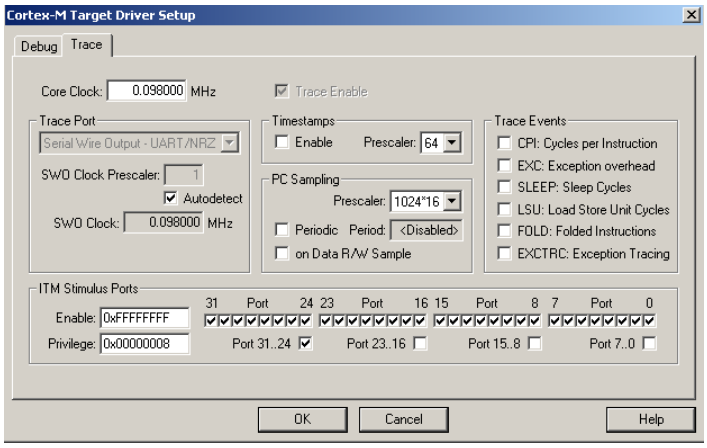


Figure 24: Trace Configuration for printf

6. Click on View/Serial Windows and select Debug (printf) Viewer.
7. Make sure Periodic Update is enabled or the window will only update when you stop the program.
8. Click on RUN.
9. In the Debug (printf) Viewer you will see the value "5" appear every few seconds as shown in Figure 25. You could add more ASCII characters to format your own messages.

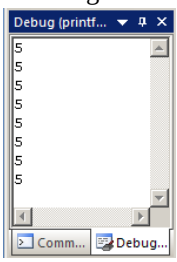


Figure 25: ITM writes of "5" with LF and CR

Trace Records

These writes to the ITM will be displayed in the Trace Records window.

ITM Conclusion

The writes to ITM Stimulus Port 0 are intrusive and are usually one cycle. It takes zero additional CPU cycles to get the data displayed in the Debug (printf) Viewer.

TIP: ITM_SendChar is a Keil supplied function you can use to send characters to the Debug Viewer. It is found in the header core_CM3.h in the directory C:\Keil\ARM\RV31\INC.

Copyright © 2010 ARM Ltd. All rights reserved

11) Opening and Using RTX Kernel Awareness: The RTX Viewer

Keil has its own full feature RTOS called RTX. It is provided as part of Keil MDK full tool suite for no charge and can have up to 255 tasks. No royalty payments are required.

If the RTX source code is required, this is included in the Keil RL-ARM™ Real-Time Library which also includes USB, CAN, TCP/IP networking and a Flash File system. µVision will work with any RTOS.

Users often want to know the number of the current operating task and the status of the other tasks. This information is usually stored in a memory area by the RTOS. Keil provides Task Awareness windows for RTX.

µVision's RTX Viewer is a kernel awareness feature with live update. This uses both the Serial Wire Viewer ITM Stimulus Port and the DAP (Debug Access Port).

RTX Viewer windows are available while in Debug mode by selecting Debug/OS Support. See Figure 23.

An example using RTX is available for SmartFusion.
RTX is easy to configure with SmartFusion. This integration is made easy by the Cortex-M3 architecture.

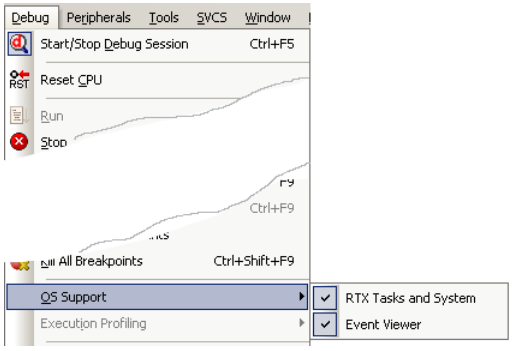


Figure 26: Opening the RTX Viewer

Configuring µVision to activate RTX Kernel Awareness:

The RTX Viewer has two components: the RTX Tasks and System window which uses the DAP for access and the Event Viewer which uses SWV (Serial Wire Viewer) through the ITM Stimulus Port # 31. Table 1 outlines the mandatory configuration items.

Table 1: µVision configuration items needed for RTX Viewer

Window\setting	**Enable RTX**	**Periodic Update**	**SWD**	**SWV**	**ITM**
RTX Tasks & System	Yes	Yes	Yes	No	No
Event Viewer	Yes	No	Yes	Yes	# 31

Notes for Table 1:

1. **Enable RTX**: Click on the Options for Target icon and select the Target tab.
   In the Operating System dialog box, select RTX Kernel – Viewer.
2. **Periodic Update**: while in Debug mode, select View/Update Periodic Window. On by default.
3. **SWD**: Serial Wire Debug: Options for Target select Debug tab. SWD is needed for SWV
4. **SWV**: Serial Wire Viewer. Options for Target select Debug tab and then Settings.
   The following items are mandatory: everything else is optional.
   a. Trace Enable must be checked and a valid entry for Core Clock (0.098 MHz for current SmartFusion silicon date code 0952).
   b. ITM Stimulus # 31.

Open RTX_Blinky and configure the RTX Viewer:

1) Open the RTX_Blinky project from C:\Keil\ARM\Boards\Actel\SmartFusion\Blinky.uvproj.
2) Ensure the SWV is configured in the Trace Configuration window. Enable ITM Port 31.
3) Compile and load the project into the Flash. Enter Debug mode.
4) Click on the RUN icon to run RTX_Blinky.

Copyright © 2010 ARM Ltd. All rights reserved

Open the RTX Viewer Windows:

1. Open Debug/OS Support and select RTX Tasks and System and the window in Figure 27 opens up. RTOS visibility is updated in real-time using the same CoreSight DAP technology as used to update and modify the Watch and Memory windows.
2. Open Debug/OS Support and select Event Viewer and the window in Figure 28 opens up.
3. Note the tabs at the bottom left to switch between the two windows.

Notes:

1. The Event Viewer is updated with the ITM Stimulus Port # 31: is part of the Serial Wire Viewer.
2. If the RTX Tasks and System window is updated only when you stop the program: check View/Update Periodic Window This setting will not affect the Event Viewer.
3. If the Event Viewer shows no data: click on the ALL icon and adjust the range with + and -. Make sure the Serial Wire Viewer is configured properly.

TIP: To view the exception SysTick, enable EXCTRC in the Trace Configuration window. Select View/Trace and select the Exception window. SysTick will also be displayed in the Instruction Trace window.

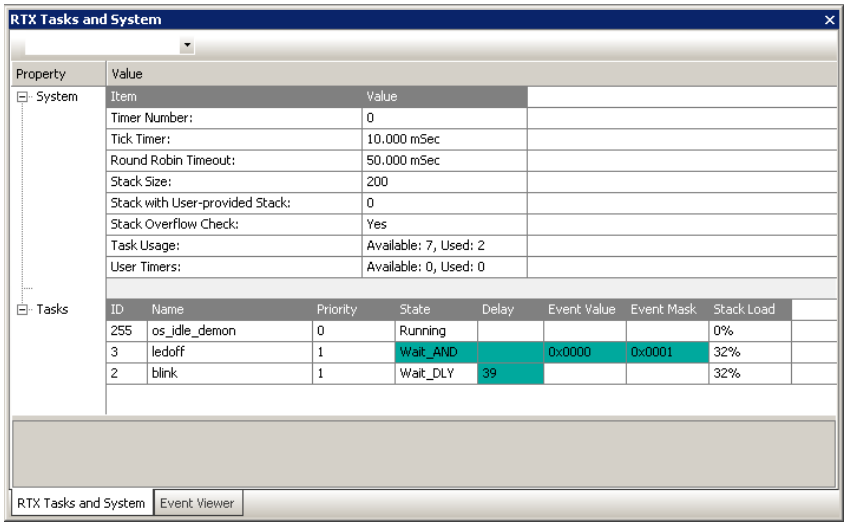


Figure 27: RTX Viewer: RTX Tasks and System window.

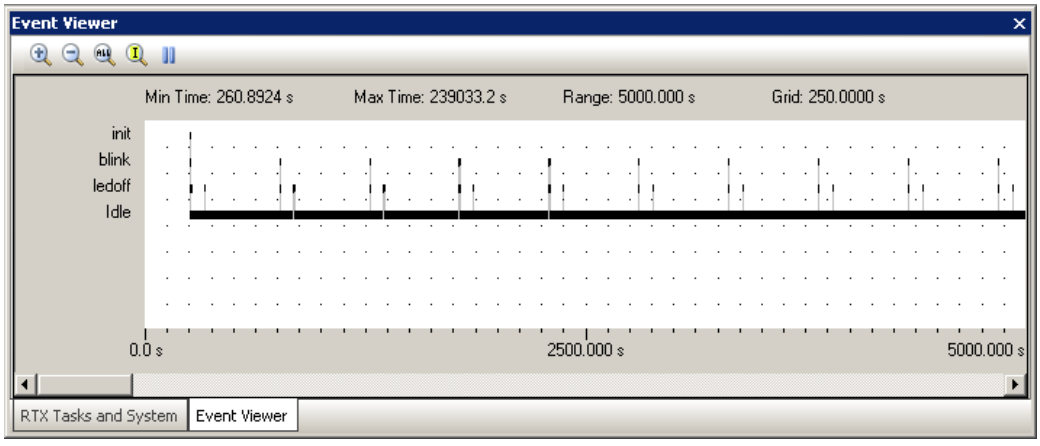


Figure 28: RTX Viewer: Event Viewer window.

Copyright © 2010 ARM Ltd. All rights reserved

12) Creating a new Task in RTX_Blinky

1. Stop the program and exit Debug mode.
2. In Blinky.c, create the global variable count near line 13: **`int count = 0;`**
3. Create the task ID of our new task: **`OS_TID t_counter;`** near line 17.
4. Create our new task called counter near line 40 between Task 2 and task 3.

```
/*----------------------------------------------------------------------------
 *        Task 2-1/2 'counter': Increment Counter
 *---------------------------------------------------------------------------*/
__task void counter (void) {
  for (;;) {
    os_evt_wait_and (0x0001, 0xffff);   /* wait for an event flag 0x0001     */
    count ++;
    os_dly_wait (8);                    /* delay 8 clock ticks               */
  }
}
```

5. Start our new task counter by placing this line in Task 2 just after the line **`GPIO->GPIO_OUT`**

```
os_evt_set (0x0001, t_counter);      /* send event to task 'counter' */
```

6. Start our Task 2-1/2 by putting this line in Task 3 just before os_tsk_delete_self (); near line 63.

```
t_counter = os_tsk_create (counter, 0);  /* start task 'counter' */
```

1. Rebuild (compile) the source files. You must have no errors or warnings displayed.
2. Load the project into the Flash . Enter Debug mode.
3. Insert the variable **`count`** into the Watch window.
4. Start the program.
5. **`count`** will start incrementing as seen in the Watch Window.
6. Open the RTX Viewer windows in Debug menu/OS Support menu.
7. Position and size these windows as necessary. Use the ALL, + and – buttons for proper scaling.
8. You will now see our new task entered and displayed in real-time as in Figure 29.

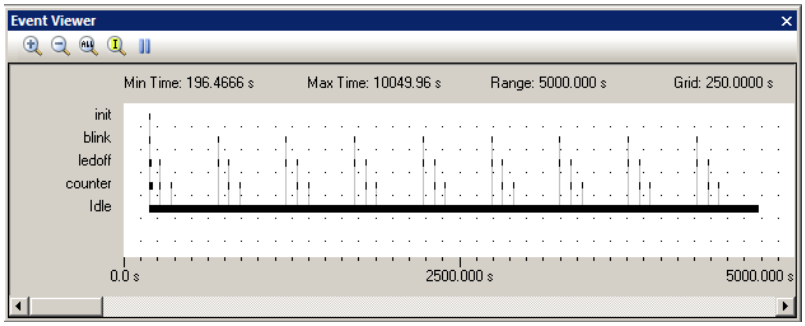


Figure 29: Event Viewer with Task counter.

TIP: You will not be able to use the Logic Analyzer window to see the variable count because 98 KHz SWO port is too slow for the data rate needed.

You can see how very easy it is to configure and use Keil RTX !

And remember…it is included free with Keil MDK…

And the sources are available with RL-ARM.

Copyright © 2010 ARM Ltd. All rights reserved

1) Creating a new project: Using the Blinky source files:

All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start this example project from the beginning to illustrate how easy this process is. We will use the existing source code files so you will not have to type them in. Once you have the new project configured; you can build, load and run the Blinky example as usual. You can use this process to create any new project from your own source files created with µVision's editor or any other editor.

Create a new project called Mytest:

1. With µVision running and not in debug mode, select Project/New µVision Project.
2. In the window Create New Project go to the folder C:\Keil\ARM\Boards\Actel\SmartFusion.
3. Right click and create a new folder by selecting New/Folder. I named this new folder FAE.
4. Double-click on the newly created folder "FAE" to enter this folder as is shown below.
5. Name your project. I called mine Mytest. You can choose your own name.
6. Click on Save.
7. "Select Device for Target 1" window opens up.
8. This is the Keil Device Database® which lists all the devices Keil supports (plus some secret ones).
9. Locate the Actel directory, open it and select A2F200M3F. Note the device features are displayed
10. Click on OK.

11. A window opens up asking if you want to insert the default SmartFusion startup code to your project. Click on "Yes". This will save you a great deal of time.
12. In the Project Workspace in the upper left hand of µVision, open up the folders by clicking on the "+" beside each folder.
13. We have now created a project called Mytest and the target hardware called Target 1 with one source file startup_A2FxxxM3.s.
14. Click once (carefully) on the name "Target 1" (or twice if not already highlighted) in the Project Workspace and rename Target 1 to something else. I chose Smart as shown in Figure 30. Press Enter to accept this. Note the Target selector also changes to Smart. Click on the + to open up the directory structure if necessary. You can create many target hardware configurations including a simulator and easily select them.

Select the source files:

1. Using MS Explore (right click on Windows Start icon), copy blinky.c, core_cm3.c and system_A2FxxxM3x.c from C:\Keil\ARM\Boards\Actel\SmartFusion\\Blinky to the Keil\SmartFusion\FAE folder.
2. In the Project Workspace in the upper left hand of µVision, right-click on "Smart" and select "Add Group". Name this new group "Source Files" and press Enter.
3. Right-click on "Source Files" and select **Add files to Group "Source Files**".
4. Select the file Blinky.c, core_cm3.c and system_A2FxxxM3.c and click on Add (once) and then Close. These will show up in the Project Workspace when you click on the + beside Source Files..
5. Select Options For Target and select the Debug tab. Make sure ULINK Cortex Debugger is selected. Select this by checking the circle just to the left of the word "Use:".
6. At this point you could build this project. Configure the ULINK for SW and you can program the flash and run it on your SmartFusion board.

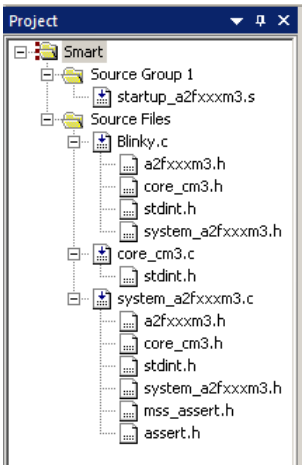


Figure 30: Project Menu

Copyright © 2010 ARM Ltd. All rights reserved

2) Running Blinky from RAM:

It is possible to run your program in the SmartFusion Cortex-M3 RAM rather than Flash.
µVision makes it easy to create different target settings. We will create one using RAM in the Blinky project.
There are six simple steps to accomplish this:

1) Create a New Target Configuration in µVision:

1. Open the project Blinky.uvproj.
2. Open Project from the main menu and select Manage and then select Components, Environment, Books…
3. In the Project Targets box, click on the Insert icon.
4. Enter the name of your target (I chose RAM) and press the Enter key. Click on OK.
5. In the main menu, click on the Select target window and select your RAM target:

6. You can now configure the target RAM which will be different from A2F200M3F Flash.

2) Create an initialization file to configure SmartFusion when Debug mode is entered:

7. We will use µVision to create an initialization file to configure SmartFusion to use RAM.
8. Select File/New and a blank text window will be created. Enter the following text:

```
FUNC void Setup (void) {
  _WDWORD(0x40006010, 0x4C6E55FA);      // Disable watchdog
  SP = _RDWORD(0x60080000);             // Setup Stack Pointer
  PC = _RDWORD(0x60080004);             // Setup Program Counter
  _WDWORD(0xE000ED08, 0x20000000);      // Setup Vector Table Offset Register
}
LOAD %L INCREMENTAL                     // Download
Setup();                                // Setup for Running
g, main
```

9. Select File/Save As and enter ram.ini and press the Enter key.

3) Enter the memory addresses into µVision:

10. Open the Target Options menu. The Target tab will be selected as in Figure 31.
11. Read/Only Memory Areas configures the program space. Read/Write Memory Areas allocates data.
12. We will divide up the SmartFusion RAM (64K in this case) between the program and data spaces.
13. RAM in the SmartFusion we are using starts at 0x2000 0000 and ends at 0x2001 0000. (64 K bytes)
14. Enter ROM Start 0x2000 0000 and size 0x8000 as shown in Figure 31. Enter RAM Start 0x2000 8000 and size 8000 as shown in Figure 31. Do not click on OK yet.

Figure 31: Configuring Memory Area (partial screen)

15. 32K of RAM is now available for your program code and 32 K of RAM for data. You can use other values. You must make sure there is enough RAM for your program and variables to fit into.

Copyright © 2010 ARM Ltd. All rights reserved

4) Create a directory for the compiled files to be stored:

16. Now we will create a directory for the compiled files. Select the Output tab.
17. Click on Select Folder for Objects… box. Select Create New Folder icon. New Folder is entered.
18. Rename it RAM and press Enter.
19. Double-click on this new folder to add it in the Path box. Click on OK. Your compiled files will go here.

5) Select ram.ini to be executed when Debug mode is entered:

20. Select the Debug tab. Options for Target should still be open. Use if you closed it.
21. In the Initialization File: box, enter ram.ini as shown in Figure 32. You can use the Browse icon.
22. ram.ini will be executed every time you enter Debug mode.
23. Unselect Load Application at Startup: This makes sure your program isn't loaded twice. Run to main() will be grayed out. This function will be done by the last command in the ram.ini file.
24. Click on Utilities and uncheck Update Target before Debugging to turn off automatic Flash programming when entering Debug mode.
25. Click OK. Select File/Save All.

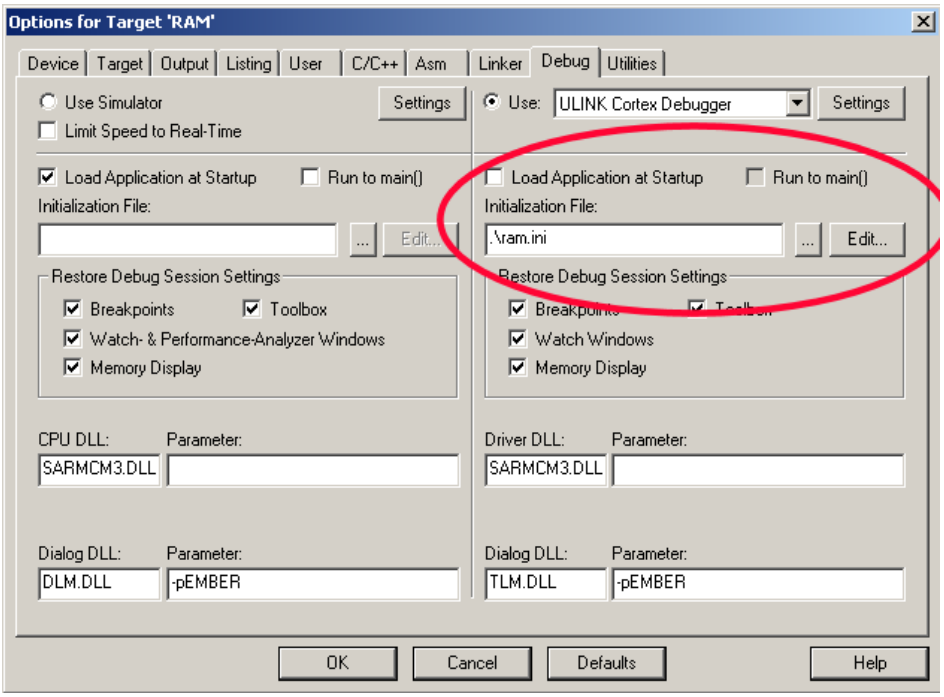


Figure 32: Selecting ram.ini

6) Compile and RUN the program:

26. Rebuild and enter Debug mode. Do not use Load. Load is only for programming Flash.
27. Note the disassembly window will show an address in the area of 0x2000 0000.
28. This is the Blinky program located in RAM.
29. Breakpoints set will be software and can be as many as you want. You cannot set them on-the-fly as with hardware breakpoints. You must stop the program to set/unset them.

Copyright © 2010 ARM Ltd. All rights reserved

3) SmartFusion eNVM Programming:

µVision provides an integral FLASH programming feature. The source files are in C:\Keil\ARM\Flash\A2FxxxM3

Configuring Flash Programming:

1) Click on the Target Options icon. You must be in Edit and not in Debug mode.
2) Click on the Utilities tab. The rear window of Figure 33 opens up.
3) Click on Settings. The front window of Figure 33 opens up.
4) Under description is the Flash programming algorithm. Figure 33 is correct.
5) If this field is blank, you will receive an error when you try to program the SmartFusion eNVM.
6) If it is blank or you need to make a change, click on Add. Select the appropriate algorithm and select OK.
7) Click on OK twice to return to the main menu.

TIP: To program the flash when entering Debug mode, select Update Target before Debugging as shown in the rear Figure 33. If not selected, you will need to program it manually with the Load icon.

TIP: The setting Verify Code Download in the Target Options window/Settings: must be deselected for SmartFusion eNVM programming. Verify is now performed by the programming algorithm. If Verify Code Download is enabled, an error will result because it will test the Actel bootloader code rather than the user flash.

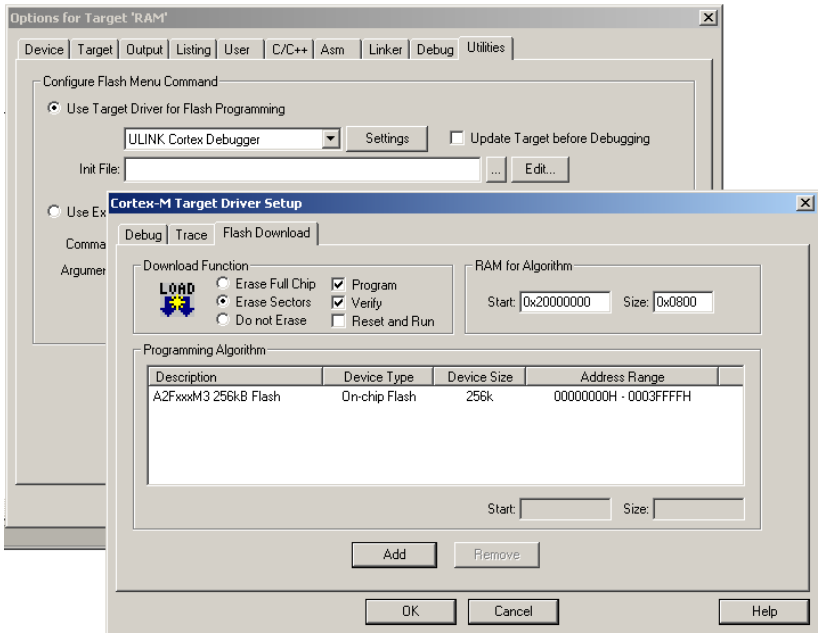


Figure 33: Flash Programming Configuration

SmartFusion BootLoader: SmartFusion contains a bootloader in its eNVM (Flash). At RESET this bootloader runs first and then control is passed to the user program typically located at 0x0.

With Blinky loaded and in Debug mode, click on the RESET icon.

Note the addresses in the disassembly window are in the area of 0x6000 0000. Probably at 0x6008 051C which is the start of the bootloader code.

Run and stop the program. Now the addresses are in the user code in the area of 0x0. This is because the Flash has now been remapped by the bootloader.

Copyright © 2010 ARM Ltd. All rights reserved

4) TCP/IP: Keil http demonstration:

Keil has ported its TCP/IP stack to the SmartFusion Ethernet peripheral. This TCP/IP is part of the Keil RL-ARM software suite.

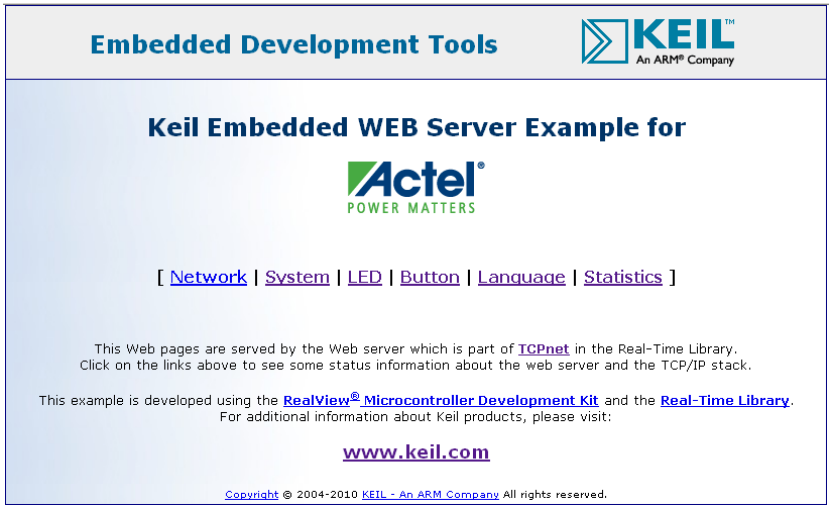
A web server (http) demonstration is available on: **www.keil.com/download/docs/404.asp**

Instructions are included as well as a pre-compiled sample executable. (Http_demo.axf).

Do not attempt to rebuild the project unless you have RL-ARM installed. Http_demo.axf will be deleted and no new version will be created. You must then reload Http_demo.axf from the example files again.

You can load this axf file into the evaluation version of µVision and run and debug the http program.

A http demon will be installed on the SmartFusion and you can connect to it with your favourite browser as shown:


Embedded Development Tools
KEIL
An ARM® Company
Keil Embedded WEB Server Example for
Actel
POWER MATTERS
[Network | System | LED | Button | Language | Statistics]
This Web pages are served by the Web server which is part of TCPnet in the Real-Time Library.
Click on the links above to see some status information about the web server and the TCP/IP stack.
This example is developed using the RealView® Microcontroller Development Kit and the Real-Time Library.
For additional information about Keil products, please visit:
www.keil.com
Copyright © 2004-2010 KEIL - An ARM Company All rights reserved.

Copyright © 2010 ARM Ltd. All rights reserved

5) ARM JTAG/SWD/SWO Adapter Connector Schematics:

For the complete description of CoreSight and the new Hi-Density debug connector search www.arm.com for this document: DDI0314F_coresight_component_trm.pdf. Many customers are now using this new, smaller connector.

Hi-Density connector part numbers can be found here: www.samtec.com/ftppub/pdf/ftsh_mt.pdf.

The 10 pin part number that Keil uses in its boards is Samtec FTSH-110. A 20 pin is available: FTSH-110.

TIP: TRSTN resets the JTAG module inside the processor. Most better debug adapters do not need it.

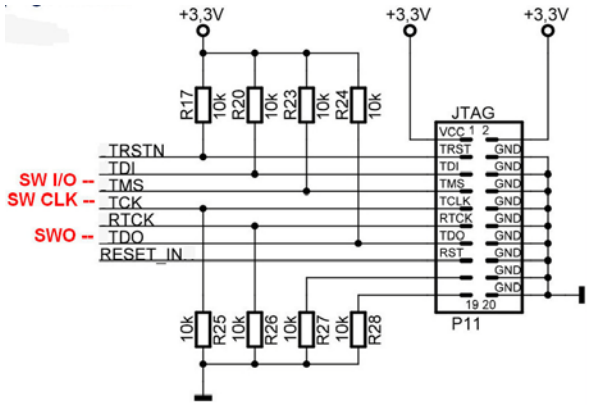


Figure 34: Legacy JTAG Connector showing location of the Serial Wire signals. Top view.

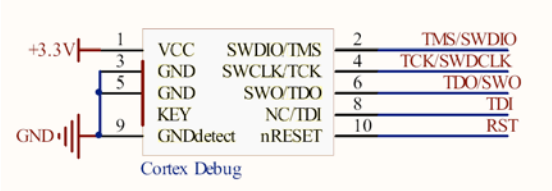


Figure 35: 10 pin Hi-Density JTAG/SWD/SWO connector. Top view.

The new ULINK-ME:

The current ULINK-ME provides 3.3 volts to a target board through JTAG connector pin 1. You can remove this feature by cutting the pin 1 lead or remove the diode on the ME. This diode is on the bottom right corner of the ME beside the JTAG connector. This will not affect the EVAL-KIT. The DEV-KIT is affected more with its complicated power supplies. The ULINK2 or ULINK*pro* are not affected as neither provides power to a target.

The new ULINK-ME also incorporates a connector and cable as shown in Figure 35 for the new compact 10 pin JTAG/SWD connector. It does not supply the 3.3 volts to pin 1. It is due for release in the summer of 2010.

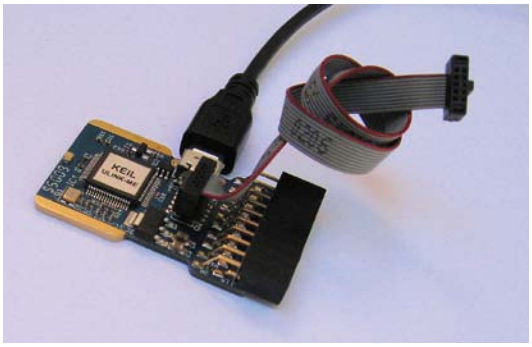


Figure 36: ULINK-ME with the new 10 pin connector and cable.

Copyright © 2010 ARM Ltd. All rights reserved

6) Acronym List

BP Break Point
DAP Debug Access Port
eNVM Actel Flash - embedded nonvolatile memory
ETM Embedded Trace Macrocell
GPIO General Purpose Input/Output
ITM Instrumentation Trace Macrocell
SWD Serial Wire Debug
SWV Serial Wire Viewer
SWO Serial Wire Output
TPIU Trace Port Interface Unit
WP Watch Points. They are also called Access Breaks.

Macrocell: A macrocell is a pre-built "black box" of gates used by the chip designer to add a function or peripheral to a processor. Examples are ETM, ITM and memory. Macrocells can also be thought of as "building blocks". This is a term used by chip designers and not end users.

Hardware Design: This is Actel's terminology for the FPGA program. Similar to a microcontroller project.

7) Overloading the SWO pin:

The Serial Wire Output pin (SWO) is just that – a single bit high speed line that is expected to handle all the data sent to it. A Cortex-M3 can read and write data much faster than the SWO can process.

Overloads have an X in the Ovf or Dly (overflow or Delay) column in the Trace Records window: but not always. Sometimes the frames have corrupt, missing or have wrong information and this is easy to spot.

Here are some hints on reducing traffic to minimize SWO overruns:

1. Reduce the number of variables in the Logic Analyzer.
2. Reduce the number of items selected in the Trace Configuration window: These include:
    a. PC Sampling. Turn this off or increase the Prescaler to reduce the sampling rate.
    b. On Data R/W Sample.
    c. EXCTRC: Exception Tracing
    d. Any Trace Events. (the Counters)
    e. ITM Stimulus Ports 31 and 0. (Viewer and RTX Awareness respectively)
3. Set the Timestamp Prescaler to 1.
4. Turn the timestamps off. Sometimes this will turn the trace off though. But it is worth a try.
5. Lower the rate of reads, writes and exception calls in your software.
6. Turn off the SYSTICK timer in your software if you are not using it.
7. Reduce the Range setting in the Logic Analyzer. Sometimes a very large number overloads µVision.
8. The General Rule is: activate only those SWV features you need.
9. Using a faster computer can help in some cases. SWV can create a great deal of information that must be processed quickly. This is especially true where the Logic Analyzer window is concerned.
10. We will be making continuous improvements to µVision to provide you with a better SWV experience.

TIP: If you are seriously overloading the SWO, there will usually be at least some data coming out. µVision recovers easily and painlessly from data overruns. You are made aware when frames are lost.

Copyright © 2010 ARM Ltd. All rights reserved

8) Top Six Reasons why you can't get SWV working:

Symptoms can range from an inability to enter a known variable in the Logic Analyzer, corrupted trace frames or no information at all.

1. Core Clock is wrong. This is the number 1 reason. Future silicon of SmartFusion will have a higher SWO speed than the current 98 KHz. Normally, the SWO speed is derived from the Core Clock speed. For the next version of SmartFusion devices, you would enter 100 MHz or whatever your clock speed is in Core Clock: and the SWO speed will be calculated from that.

   With silicon code date 0952 calculate Core Clock speed as (CPU Speed/1024). 100 MHz/1024 = 98 KHz.
2. Periodic Update is not enabled. Your windows update only when the processor is stopped.
3. Trace Enable is not checked.
4. Using JTAG instead of SWD. µVision will complain about this if you try it.
5. SWD speed too high. Try a slower speed. This is not usually the problem.
6. You have selected too many SWV items and it is extremely seriously overloaded. This effect often manifests itself when you stop the program and the Trace Records window is updated.
7. Certain portions of µVision in MDK 4.11 are in the process of being updated. This is particularly true of the Logic Analyzer and ULINK*pro* features. With the current limited bandwidth of the SmartFusion SWO pin, a few Serial Wire Viewer features might not work consistently as described. These will be updated as soon as possible.

For more information:

Contact Keil or your local representative as listed on www.keil.com/distis

Obtain the Keil ULINK*pro* User Manual for information on Serial Wire Viewer.

Visit www.keil.com for the latest version of this document.

For Actel: www.actel.com/products/SmartFusion

For additions, comments or corrections to this document: contact Robert Boys bob.boys@arm.com

Copyright © 2010 ARM Ltd. All rights reserved